UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS AGRÁRIAS E VETERINÁRIAS
CÂMPUS DE JABOTICABAL

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE MODELOS PARA SIMULAÇÃO DE DADOS SNPs

Adam Taiti Harth Utsunomiya
Zootecnista

JABOTICABAL – SÃO PAULO – BRASIL
2010

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS AGRÁRIAS E VETERINÁRIAS
CÂMPUS DE JABOTICABAL

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE MODELOS PARA SIMULAÇÃO DE DADOS SNPs

Adam Taiti Harth Utsunomiya

Orientador: Prof. Dr. Ricardo da Fonseca

Dissertação apresentada à Faculdade de Ciências Agrárias e Veterinárias – Unesp, Câmpus de Jaboticabal, como parte das exigências para a obtenção do título de Mestre em Genética e Melhoramento Animal.

JABOTICABAL – SÃO PAULO – BRASIL
Setembro de 2010

Utsunomiya, Adam Taiti Harth
U92d Desenvolvimento e implementação de modelos para simulação de dados SNPs / Adam Taiti Harth Utsunomiya. – – Jaboticabal, 2010
iv, 63 f. : il. ; 28 cm

Dissertação (mestrado) - Universidade Estadual Paulista, Faculdade de Ciências Agrárias e Veterinárias, 2010
Orientador: Ricardo da Fonseca
Banca examinadora: Lúcia Galvão de Albuquerque, Lenira El Faro
Bibliografia

1. Marcador molecular 2. SNP. 3. Simulação. 4. Software. I. Título. II. Jaboticabal-Faculdade de Ciências Agrárias e Veterinárias.

CDU 636.082

Ficha catalográfica elaborada pela Seção Técnica de Aquisição e Tratamento da Informação – Serviço Técnico de Biblioteca e Documentação - UNESP, Câmpus de Jaboticabal.

CERTIFICADO DE APROVAÇÃO

DADOS CURRICULARES DO AUTOR

ADAM TAITI HARTH UTSUNOMIYA – nasceu aos 22 de maio de 1985, na cidade de São Carlos – SP. Cursou do ensino primário até ao 5º ano do ensino fundamental na cidade de Campinas – SP e o restante do ensino fundamental e médio na cidade de Araçatuba-SP. Em 2003, iniciou o curso de zootecnia na Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP – *Campus* de Dracena – SP, concluindo-o em julho de 2008. Em agosto de 2008, ingressou no curso de mestrado em Genética e Melhoramento Animal pela Faculdade de Ciências Agrárias e Veterinárias (FCAV) da UNESP, *Campus* de Jaboticabal, sob orientação do Prof. Dr. Ricardo da Fonseca.

“A melhor de todas as coisas é aprender. O dinheiro pode ser perdido ou roubado, a saúde e força podem falhar, mas o que você dedicou à mente é seu para sempre.”

Louis L’Amour

Aos meus bisavós Afonso Harth e
Palmira Larroyel Harth,
“In memorian”,
Dedico

AGRADECIMENTOS

Agradeço aos meus pais Takashi Utsunomiya e Josiane Cristina Harth Scanzani por todo o suporte oferecido a mim.

Ao Doutor Ricardo da Fonseca, que sempre esteve disposto a me orientar e que foi mais do que um professor e orientador, um amigo que teve a paciência de compreender-me em momentos que até mesmo eu não me compreendia.

Aos amigos Rodrigo Vaz, Rodrigo Marques, Aline Buda, Mirella Vulcano, Susana Cazerta, Fernanda Grano, André Fernando e Bruno Fonseca por tantos anos de alegria e amizade dedicada. Vocês fizeram parte desta conquista.

Ao grande amigo Michel Farah, que em tão pouco tempo de amizade demonstrou o que é ser um amigo, se dispondo a ajudar em todos os momentos, bons e ruins.

Aos amigos Rodrigo Longuine e Mclean Mcnabb, pelos momentos de descontração.

Aos amigos do LuCCA-Z, Alexandre, Rafael, Michele, Daniel e Orlando. Todos colaboraram para que este trabalho se concluísse.

Por fim, todos àqueles que fizeram e fazem parte da minha vida, os meus agradecimentos.

SUMÁRIO

DESENVOLVIMENTO E IMPLEMENTAÇÃO DE MODELOS PARA SIMULAÇÃO DE DADOS SNPs

RESUMO – Um grande volume de informações de marcadores SNPs vem sendo produzidos e aplicados a metodologias de avaliação genética animal. A quantidade de informações disponíveis faz-se um desafio, pois armazená-las e processar-las é demandante computacionalmente. Então, propôs-se com este trabalho 2 algoritmos (MLOCO e MSNP) para simular dados SNPs como alternativa de minimizar a demanda por processamento e consumo de memória computacional. MLOCO simula SNP como um loco que possui configuração de alelos, posição no genoma e efeitos sobre a expressão de fenótipos (nulos para SNP). MSNP simula SNP apenas como loco que possui configuração de alelos e posição no genoma. Ao nível de cromossomo, para MLOCO, poligenes, QTLs e SNPs são armazenados em um único vetor de acordo com suas localizações. Para MSNP, poligenes, QTLs e SNPs também são armazenados de acordo com suas localizações, porém em vetores específicos para cada tipo de loco. Considerando o consumo de memória, MLOCO é menos eficiente porque armazena um número de variáveis maior (efeito do loco, mesmo que seja zero). MSNP não armazena esta variável. Quanto à velocidade de processamento, MLOCO é mais eficiente porque os locos são armazenados de maneira seqüencial, facilitando a amostragem dos alelos devido a variável “posição”, para formação de um gameta e consequentemente um indivíduo. Como o fator limitante na realização de simulações e utilizações de dados é a memória RAM, o MSNP é a melhor alternativa para simular dados SNPs.

Palavras-chave: marcador molecular, SNP, simulação, software.

DEVELOPMENT AND IMPLEMENTATION OF MODELS FOR SNPs DATA SIMULATION

SUMMARY – A large amount of information of SNPs markers have been produced and applied methodologies in genetic evaluation. The amount of information available makes it challenging, for storing and processing them is computationally expansive. So, it was proposed in this paper two algorithms (MLOCO and MSNP) to simulate data SNPs as an alternative to minimize the demand for processing and consumption of computer memory. MLOCO simulates SNP as a *locus* that has configuration of alleles, position in the genome and its effects on the expression phenotypes (null for SNP). MSNP SNP simulates only *locus* that has position and configuration of alleles. At the level of the chromosome to MLOCO, polygenes, QTLs and SNPs are stored in a single vector according to their locations. To MSNP, polygenes, QTLs and SNPs are also stored according to their locations, but in specific vectors for each *locus* type. Whereas the memory consumption, MLOCO is less efficient because stores a greater number of variables (*locus* effect, even if it is zero). MSNP does not store this variable. As for processing speed, MLOCO is more efficient because the loci are stored sequentially, facilitating the sampling of alleles due to the variable "position" to form a gamete and hence an individual. As the limiting factor in simulations and use data is the RAM memory, the MSNP is the best alternative for simulating SNPs data

Keywords: molecular marker, SNP, simulation, software.

1 – INTRODUÇÃO

A prática do melhoramento animal está fundamentada na seleção de reprodutores geneticamente superiores para determinada característica. Tradicionalmente, esta seleção baseia-se em informações fenotípicas associadas com informações genealógicas. Mas, com o advento das tecnologias de prospecção de *loci* de características quantitativas e marcadores moleculares, estão surgindo novas metodologias de seleção.

A "Genome Wide Selection" (GWS), proposta por MEUWISSEN *et al*. (2001), utiliza informações de marcadores "Single Nucleotide Polimorphisms" (SNPs) para predição de valores genéticos. Estes marcadores (SNPs) têm tido destaque no cenário científico mundial devido ao seu alto grau de informatividade, derivada do volume de polimorfismos encontrados nos genomas. Assim, existe uma grande probabilidade de um marcador tipo SNP estar associado com genes que contribuem de forma diferenciada na expressão de uma característica de interesse econômico. Esta probabilidade é analisada por estudos de associação do genoma completo ("Genome Wide Association Studies – GWAS").

Um dos maiores problemas enfrentados na realização de avaliações genéticas em programas de seleção é a alta demanda por memória e processamento dos computadores, devido à quantidade de informações necessárias para predizer o mérito genético dos indivíduos. A metodologia comumente aplicada para predizê-los é a resolução de sistemas de equações lineares, que requer o preenchimento de imensas matrizes de coeficientes, tão maiores quanto maior for o volume de informações disponíveis. No caso da GWS, as informações de SNPs preencheriam uma destas matrizes, o que, computacionalmente, demandaria bastante tempo de processamento e memória para armazenamento dos dados, necessitando de equipamentos sofisticados ("hardware") e programas de alto desempenho ("software").

A comunidade científica tem apresentado, ao longo dos anos, seu esforço para superar as dificuldades de aplicação de metodologias que manipulam muitos dados. Algoritmos iterativos para resolução de sistemas lineares, inversão de matrizes,

simulação de dados, construção de “softwares”, entre outros, têm sido propostos. Como a prospecção de SNPs aumenta continuamente e os custos com genotipagem ainda são um fator limitante no desenvolvimento de pesquisas, um trabalho que aborde o desenvolvimento e implementação de modelos que permitam simular SNPs pode trazer benefícios.

Este trabalho propôs desenvolver e implementar 2 modelos para simulação de SNPs.

1.1 – REVISÃO DE LITERATURA

A expressão de características de interesse econômico na produção animal é governada por muitos genes, características quantitativas. FISCHER (1918) *apud* HAYES (2007) propôs o “modelo infinitesimal”, o qual afirma que as características de produção são determinadas por um número infinito de *loci* não associados, que contribuem, igualmente, com efeito genético aditivo. HAYES e GODDARD (2001) refinam o modelo de FISCHER (1918) mostrando que existem poucos genes que exercem grande influência em características quantitativas e muitos genes que oferecem pequena contribuição aditiva a elas.

Uma estratégia criada e aplicada para identificar, a nível molecular, variações em características quantitativas é o mapeamento de QTL. Este é baseado na associação de regiões cromossômicas com variações fenotípicas.

De maneira geral, genes envolvidos na expressão de características de interesse são desconhecidos. Neste caso realizam-se estudos de associação entre marcadores moleculares e expressões fenotípicas.

Os marcadores são entidades físicas capazes de diferenciar, de forma inequívoca, dois ou mais indivíduos entre si. Eles podem ser marcadores morfológicos, bioquímicos ou de DNA. Os dois últimos são, comumente, conhecidos como marcadores moleculares.

VIGNAL *et al*. (2002) propuseram duas classificações para os marcadores moleculares de importância histórica na produção animal: uma baseada em sua natureza bioquímica e outra de acordo com a natureza dos alelos. Na primeira classificação os marcadores estão divididos em: (1) inserções e deleções (Indels); (2) polimorfismo de nucleotídeo simples (SNP); e (3) número variável de repetições em tandem (VNTR) não-codificantes. Na segunda eles são classificados como: (1) bialélico dominante; (2) bialélico co-dominante; e (3) multialélico co-dominante. De maneira geral, apenas dois tipos de marcadores são explorados na produção animal: VNTR, representado pelos microssatélites, e SNPs.

Os marcadores SNPs tem como base as alterações mais elementares da molécula de DNA, ou seja, mutações em bases únicas da cadeia de bases nitrogenadas (Timina, Adenina, Citosina e Guanina), exemplificado pela Figura 1. As mutações mais comuns são as transições, onde ocorrem trocas de uma purina por outra (A→G ou G→A) ou uma pirimidina por outra (C→T ou T→C). Menos frequentes, as transversões ocorrem quando há troca de uma purina por uma pirimidina (A→T ou G→C), ou vice-versa (T→A ou C→G) (CAETANO, 2009).

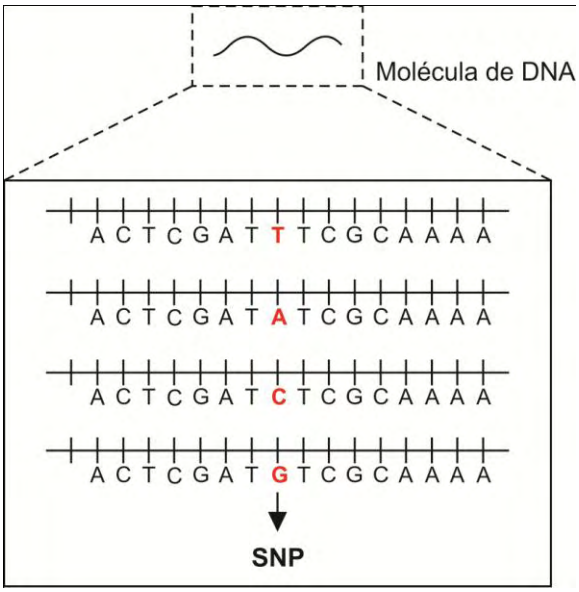

Figura 1. Representação de polimorfismo de um único nucleotídeo (SNP).

Tipicamente, os SNPs são bialélicos e co-dominantes. Isto significa que eles possuem apenas duas variantes alélicas, ambas de igual influência. Segundo CAETANO (2009), os SNPs podem ocorrer em regiões codificadoras ou com função regulatória, porém, na maioria das vezes, são encontrados em espaços intergênicos, sem função determinada. De acordo com KIM e MISRA (2007), SNPs em regiões não-codificantes não alteram a formação de proteínas (em geral), mas servem como importantes marcadores físicos nos estudos de genômica evolutiva e comparativa.

Algumas técnicas de identificação de marcadores moleculares, anteriormente descritas como os próprios marcadores, tais como “Random Amplification of Polimorphic DNA – RAPD”, “Amplified Fragment Length Polimorphism – AFLP” e “Restriction Fragment Length Polimorphism – RFLP” identificam polimorfismo de um único nucleotídeo em sua base molecular. No entanto, estes métodos são laboriosos e limitados, o que tem levado a descontinuação de seu uso (CAETANO, 2009).

Existem diversas metodologias para identificação de SNPs, tais como “primer extension”, hibridização, ligação, clivagem enzimática, espectrometria de massa e detecção de sinais baseados em fluorescência e quimiluminescência (KIM e MISRA, 2007). Uma das técnicas mais aplicadas para detecção destes marcadores é o seqüenciamento dos produtos da PCR utilizando o método de seqüenciamento Sanger (SANGER *et al*., 1977), destinado a uma região específica do genoma. Os resultados obtidos por esta técnica são contrastados com uma sequência referência. Contudo, metodologias de genotipagem de SNPs baseadas em microarranjos de DNA fabricados através da impressão de oligonucleotídeos em lâminas de vidro (HACIA *et al*., 1999) é que alavancaram a produção de centenas de milhares de marcadores.

A companhia Illumina® desenvolveu em 2010, em colaboração com a USDA ARS, University of Missouri e University of Alberta, o BovineSNP50 BeadChip (Figura 2. a), um microarranjo contendo cerca de 54.000 SNPs, e em colaboração com a USDA ARS, UNCEIA INRA, Pfizer Animal Genetics e a University of Missouri, o BovineHD BeadChip (Figura 2. b), contendo cerca de 777.000 SNPs. Ambos os microarranjos detectam marcadores igualmente espaçados pelo genoma bovino.

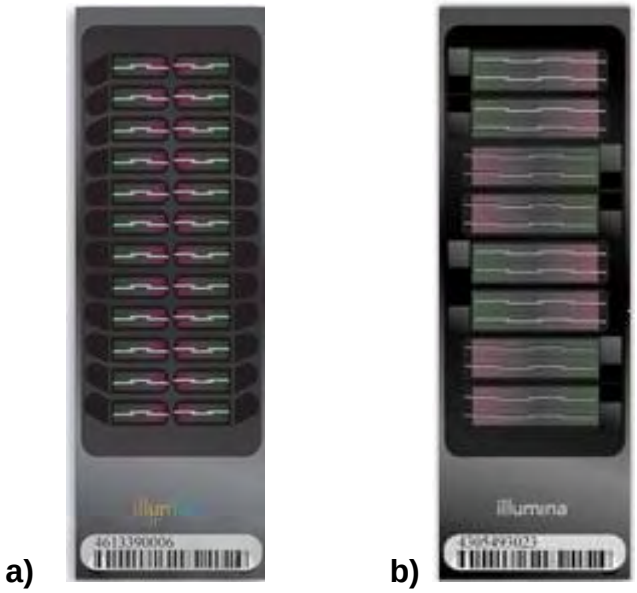

Figura 2. a) Ilustra a plataforma v2 (24 amostras em paralelo) do BovineSNP50 BeadChip da Illumina® e b) Ilustra o BovineHD BeadChip (8 amostras em paralelo) da Illumina®.

É notável o esforço da comunidade científica para identificar SNPs e associá-los a expressões de características de interesse econômico na produção animal. VAN TASSEL *et al*. (2008) contribuíram com a identificação de aproximadamente 23.000 SNPs na espécie bovina. O projeto genoma bovino permitiu a geração de grandes estudos com este tipo de marcador, tal como o "Bovine HapMap Consortium", que analisou 37.470 SNPs em 497 animais de 19 raças com diferentes biologia e distribuição geográfica, incluindo animais taurinos, zebuínos e seus cruzamentos (THE BOVINE HAPMAP CONSORTIUM, 2009). ECK *et al*. (2009) mencionaram que, durante a execução do projeto genoma bovino, foram identificados cerca de 2 milhões de SNPs na raça Hereford. Semelhantemente ao "Bovine HapMap Consortium", o "Sheep HapMap Consortium" identificou 6.021 SNPs para ovinos de 23 raças domésticas e 10 selvagens (INTERNATIONAL SHEEP GENOMICS CONSORTIUM, 2009).

Este volume de informações de SNPs tem sido aplicado em programas de seleção para predição de valores genéticos de candidatos à reprodução. HAYES *et al*. (2009) descreveram um exemplo de aplicação da informação de marcadores SNPs em programas de melhoramento para bovinos leiteiros na Austrália, Nova Zelândia, Estados Unidos e Holanda.

No cenário atual, a metodologia que prediz valores genéticos e que tem explorado informações de SNPs é a Seleção Genômica (“Genome Wide Selection – GWS”) (MEUWISSEN, HAYES e GODDARD, 2001). Os autores propõem três maneiras de predizer valores genéticos, estimando os efeitos de todos os genes associados às informações de SNPs simultaneamente. São elas: (1) Quadrados Mínimos (“Least Square - LS”); (2) BLUP (“Best Linear Unbiased Prediction”); e (3) Estimação Bayesiana. Esta metodologia, como qualquer outra que proponha utilizar um grande volume de dados, demandará por recursos computacionais de alto desempenho. Dentre eles, computadores de última geração, componentes de “hardware”, e programas sofisticados, componentes de “software”.

A construção de “softwares” para explorar diferentes aspectos da utilização de dados genômicos é evidente. Euclydes (1996) desenvolveu um sistema para simulação de dados e avaliação de métodos de seleção clássicos e associados a marcadores moleculares (GENESYS). Este sistema permitiu simular indivíduos em nível de gene, criando *loci* marcadores, tipo RFLP e microssatélites, associados à *loci* de efeito maior para estudos de avaliação genética.

Hoggart (2007) cita outras ferramentas que simulam dados genéticos de populações, tais como FPG e simuPOP, e propõem o “software” FREGENE. Estes programas foram construídos para simulação de marcadores associados a estudos de caso-controle de doenças em humanos.

O *software* estatístico R, parte oficial do projeto GNU da “Free Software Foundation” (http://www.r-project.org/), também permite a simulação de dados genéticos por meio de suas funções. Por ser um “software-livre” e possuir código-fonte aberto, este pode ser modificado pelo usuário para fins específicos. Existem “packages” que trabalham com análise de dados SNPs em R, tais como o GenAbel, disponível em (http://mga.bionet.nsc.ru/~yurii/ABEL/GenABEL) e Bioconductor, disponível em (http://www.bioconductor.org).

Ignacy Miztal (http://nce.ads.uga.edu/~ignacy/genomics/) disponibilizou o “software” SNP_SIM, um simples simulador de fenótipos e genótipos SNPs. SNP_SIM não simula meiose e acasalamentos, apenas a população base. Estas peculiaridades inflexibilizam sua aplicação em estudos de herança e associação gênica, por exemplo.

Apesar de muitos programas estarem disponíveis na “World Wide Web”, não existem trabalhos que especifiquem um modelo de simulação de SNPs.

Contudo, o volume de informações de SNPs vem aumentando exponencialmente, demandando elevada eficiência no processamento dos dados e na utilização da memória do computador. Portanto, desenvolver e implementar modelos que permitam simular informações de SNPs de maneira computacionalmente eficiente e com qualidade (similaridade com dados reais), é consideravelmente importante, pois seus resultados podem intensificar as pesquisas com bioinformática em genética e melhoramento animal.

2 – PROPOSIÇÃO DOS MODELOS

2.1 – MODELO LOCO (MLOCO)

Considera-se um loco uma região do DNA, parte de um cromossomo, que codifica uma proteína, a qual pode ter maior ou menor influência na expressão de uma característica qualquer. Cada loco em um indivíduo pode apresentar dois alelos embora, considerando-se a população, mais de dois alelos podem existir para o mesmo loco. Admitindo-se apenas dois alelos na população, A_1 e A_2, e que o primeiro deles codifica para a presença de uma determinada proteína e o segundo para ausência desta, é possível dizer que cada um desses alelos tem um efeito particular sobre a expressão da característica, o que pode ser chamado de efeito médio do alelo ou efeito médio de um gene (FALCONER e MACKEY, 1996).

A soma dos efeitos médios dos alelos de um loco diz-se efeito do loco. Assim, para um único loco bialélico, três configurações de alelos são possíveis: A_1A_1, A_1A_2 e A_2A_2.

A cada configuração está associado um efeito de loco. Se o loco é pleiotrópico ele tem efeito em mais de uma característica, sendo que esses não são necessariamente de mesma magnitude, ou seja, na característica C_1 o efeito é X_1 e na

característica C_2 é X_2, sendo que $X_1 \neq X_2$. Esse fato pode ocorrer porque a proteína codificada pelo loco pode ser fundamental em uma via metabólica associada a C_1 e ter papel secundário em uma via metabólica associada a C_2. Nesse caso, $X_1 > X_2$.

Assim, sob o ponto de vista da simulação computacional, um loco pode ser definido como um conjunto de informações:

$$L = \{x_i : i = 1, 2, 3 \ldots z\} \qquad (1)$$

Por exemplo, x_1 = configuração dos alelos, x_2 = posição do loco no cromossomo, x_3 = efeito do loco para a característica 1, entre outras.

O modelo sugerido é ilustrado na Figura 3.

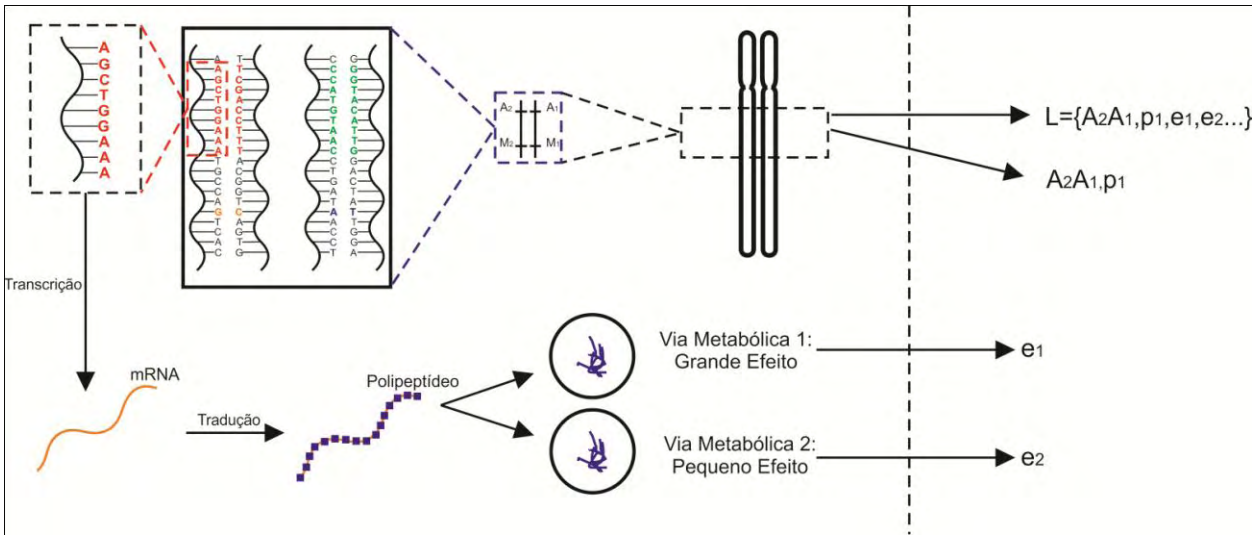

Figura 3. Ilustração do conjunto de informações que definem um loco.

Os SNPs são polimorfismos de nucleotídeo simples, ou seja, são variações ao nível de um único nucleotídeo. Os nucleotídeos não codificam proteínas e, portanto os SNPs não possuem efeitos, sendo utilizados somente como marcadores moleculares. Entretanto, a configuração das bases púricas e pirimídicas nos cromossomos homólogos define alelos (Figura 4).

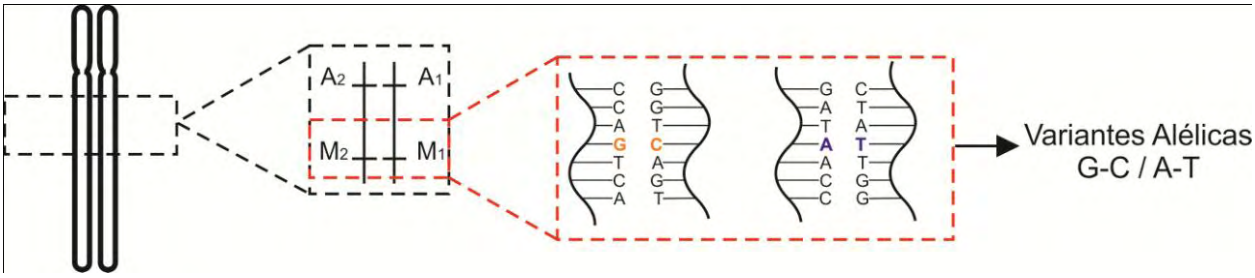

Figura 4. Variantes alélicas para polimorfismos de nucleotídeos simples (SNPs).

O modelo de loco ou variantes com pequenas modificações são frequentemente utilizados nas simulações ao nível de gene para a geração de locos poligênicos e locos de grande efeito. Assim, seria muito útil utilizar o mesmo modelo para a simulação de SNPs, enxergando-os como um loco qualquer, porém sem efeitos.

Essa técnica é possível se considerarmos que o conjunto L terá valores nulos para alguns de seus elementos:

$$L = \{x_i : i = 1, 2, 3 \dots z, x_1, x_2 \dots x_t \neq 0 \Rightarrow t < z\} \qquad (2)$$

A Figura 5 mostra o modelo de MLOCO utilizado para a simulação de SNPs.

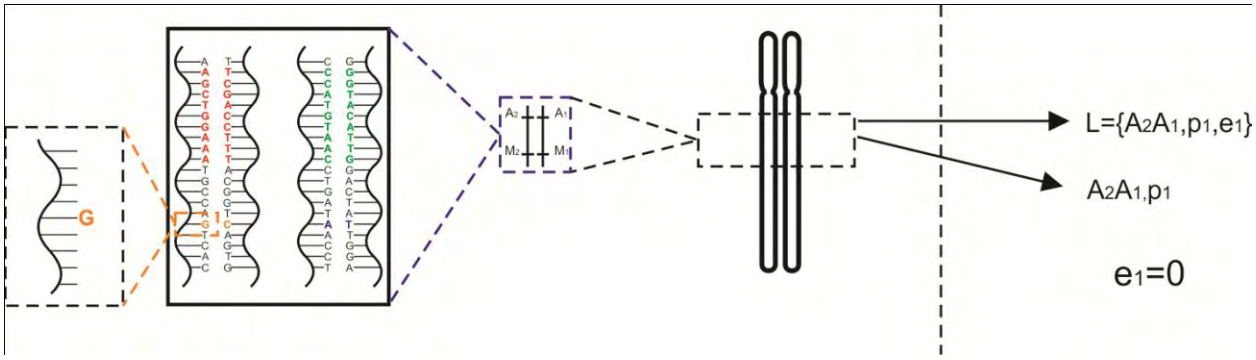

Figura 5. Ilustração do conjunto de informações que definem o modelo MLOCO para simulação de SNPs.

O SNP simulado como um loco conterá a mesma quantidade de informação, z, de um loco poligênico ou de grande efeito, no entanto, uma parte delas, z-t, receberá o valor 0.

2.2 – MODELO SNP (MSNP)

Uma vez que para a simulação de SNPs somente poucas informações do conjunto L são necessárias, é natural formular um modelo que conduza a simulação de SNPs com somente as informações necessárias para essa finalidade.

Matematicamente, podemos representar o modelo como um subconjunto de L, ou seja:

$$S \subset L \Rightarrow S = \{x_i : i = 1, 2, 3 \ldots t, t < z\} \qquad (3)$$

A Figura 6 mostra a representação gráfica desse modelo.

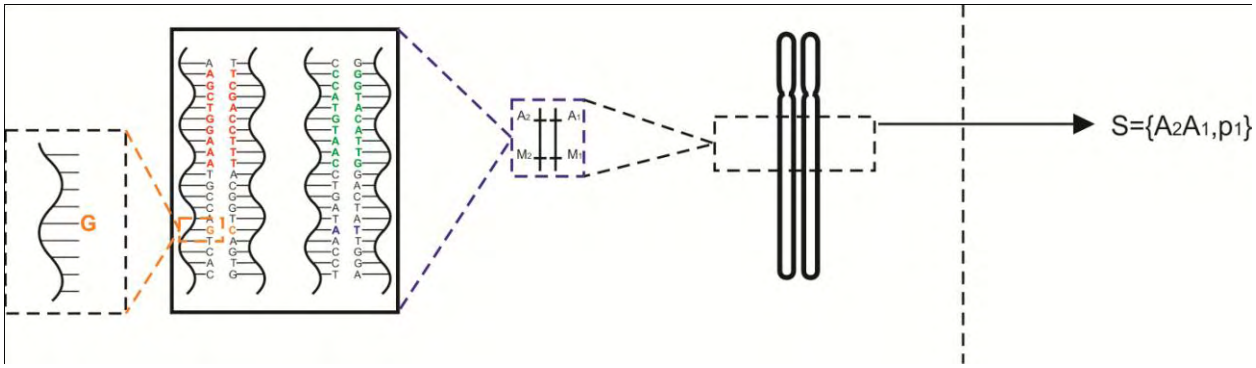

Figura 6. Ilustração do conjunto de informações que definem o modelo MSNP.

3 – COMPARAÇÃO ENTRE MLOCO E MSNP

A proposição de modelos tem dois objetivos:

1. Definir a quantidade e a qualidade da informação a ser utilizada para a execução de uma tarefa; e
2. Otimizar a utilização da memória RAM e das estruturas de processamento do computador.

Dessa forma, a comparação de modelos concorrentes deve levar em consideração esses dois aspectos.

Considere que se deseja executar uma simulação em que n indivíduos devem ser gerados com informação de uma característica qualquer, determinada por l locos poligênicos. Adicionalmente, para cada indivíduo da população s SNPs devem ser gerados.

3.1 – UTILIZAÇÃO DA MEMÓRIA RAM

Considere o modelo 1 como mostrado na seção 2.1 e defina o tamanho de L como t_1, ou seja:

$$L = \{x_i : i = 1, 2, 3 \dots t_1\}$$

Assumindo que cada informação, x_i, ocupa uma unidade de memória (u), um loco poligênico ou um SNP, simulado de acordo com esse modelo, ocuparia $t_1 x u$ unidades de memória.

Embora para SNPs algumas das informações utilizadas para simulação de locos poligênicos não sejam necessárias, de acordo com o modelo proposto, essas devem ser simuladas e recebem valor nulo.

Para cada indivíduo simulado $l+s$ conjuntos L devem ser criados e, portanto, $(l+s)t_1$ unidades de memória serão utilizadas. Se n indivíduos devem ser simulados, a quantidade de total de memória RAM utilizada empregando-se o modelo 1 será $n(l+s)t_1$.

Considere agora que os SNPs serão simulados de acordo com o modelo proposto na seção 2.2, e que o tamanho do conjunto S é t_2:

$$S = \{x_i : i = 1, 2, 3 \ldots t_2\}$$

Assim, os locos poligênicos seriam simulados de acordo com o modelo 1 e ocupariam, cada um, t_1 unidades de memória. Os SNPs seriam simulados de acordo com o modelo 2 e ocupariam, cada um, t_2 unidades de memória.

Um indivíduo contém l conjuntos L e s conjuntos S e, portanto, para ser simulado, utilizará lt_1+st_2 unidades de memória. Com n indivíduos simulados $n(lt_1+st_2)$ unidades de memória serão necessárias.

Uma medida de eficiência relativa dos dois modelos quanto à utilização de memória (E_m) pode ser definida pela quantidade:

$$E_m = \frac{n(lt_1+st_2)}{n(l+s)t_1} = \frac{lt_1+st_2}{lt_1+st_1} \qquad (4)$$

quanto mais próximo de zero o valor de E_m mais eficiente é o modelo 2 para economia de memória RAM. Por outro lado, quanto mais próximo de 1 for E_m, menos eficiente é o modelo 2.

De modo geral, a quantidade de SNP a ser simulado é muito maior que a quantidade de locos poligênicos, ou seja, $s > l$ e $\frac{l}{s} \cong 0$. Portanto, (4) pode ser aproximado como:

$$\lim_{l/s \to 0}(E_m) = \frac{t_2}{t_1} \qquad (5)$$

A eficiência relativa na utilização da memória RAM é função somente dos tamanhos dos conjuntos S e L. De outra maneira, pode-se dizer que o modelo será tão mais eficiente para economia de memória quanto menor for a quantidade de informação utilizada para caracterizar o SNP.

Até esse ponto, por questões de simplicidade, assumiu-se que cada informação x_i de L ou S ocupa a mesma quantidade de unidades de memória, o quê, de modo geral, não é o caso. Por exemplo, na definição dos conjuntos L e S podemos assumir que x_1 é a informação de configuração dos alelos, x_2 é a posição do loco, ou SNP, no cromossomo e x_3, que só existe em L, é o efeito do loco poligênico para a característica a ser simulada. O elemento x_1, por sua vez, pode também ser representado com um conjunto C de dois elementos, c_1 e c_2.

Os SNPs são, de modo geral, bialélicos e podem ser implementados como estados diferentes de unidade de memória (u). Para locos poligênicos é razoável se considerar alelos múltiplos e os alelos não podem ser representados como para SNPs, necessitando, por exemplo, de $8u$ para armazenamento de cada alelo.

Suponha também que x_2 (posição do loco ou SNP) possa ser armazenado em $8u$ e que x_3 necessite de $32u$ para armazenar o efeito do loco poligênico. Assim, um SNP necessita de: $2u + 8u = 10u$, em que as duas unidades de memória são devidas ao armazenamento de C, e as 8 unidades são devidas ao armazenamento da posição.

Um loco poligênico usaria $16u$ para armazenamento de C, $8u$ para a posição e mais 32 unidades para o efeito do loco na expressão da característica. Portanto, a quantidade de memória necessária é $56u$.

Assim, (4) pode ser reescrito como:

$$E_m = \frac{n(lU_1+sU_2)}{n(l+s)U_1} = \frac{lU_1+sU_2}{lU_1+sU_1} \qquad (6)$$

em que,

$U_1 =$ quantidade total de unidades de memória RAM utilizada por um loco poligênico ou SNP definido pelo modelo 1 = $\sum_{i=1}^{t_1} u_i$, onde u_i é a quantidade de unidades de memória demandada pela informação x_i.

$U_2 =$ quantidade de unidades de memória RAM utilizada por um SNP definido pelo modelo 2 = $\sum_{i=1}^{t_2} u_i$.

Como a quantidade de SNP é muito maior que a de locos poligênicos, (5) pode ser definido como:

$$\lim_{l/s \to 0}(E_m) = \frac{U_2}{U_1} \qquad (7)$$

3.2 – UTILIZAÇÃO DA ESTRUTURA DE PROCESSAMENTO

Assume-se aqui que a demanda por processamento é definida por 2 elementos:

1. Quantidade de informação a ser processada; e
2. Complexidade da informação a ser processada.

O modelo 1 define que um SNP deve ser criado como um loco poligênico, ou seja, pode ser representado por L, como definido em (1). O tamanho do conjunto L é t_1.

O modelo 2 define que o SNP deve ser criado como uma entidade particular, como o conjunto S definido em (2). O tamanho de S é t_2.

Se considerarmos que o número de passos ou de unidades de esforço de processamento (p) é o mesmo para a criação de cada informação nos conjuntos L e S, então, o tempo de processamento é proporcionalmente maior no conjunto que contém maior número de informações. Utilizando diagramas de Venn (Figura 7), o tempo de processamento adicional para a geração de um SNP, de acordo com o modelo 1, é representado pela área hachurada, ou seja, as informações adicionais estão contidas em L mas não em S.

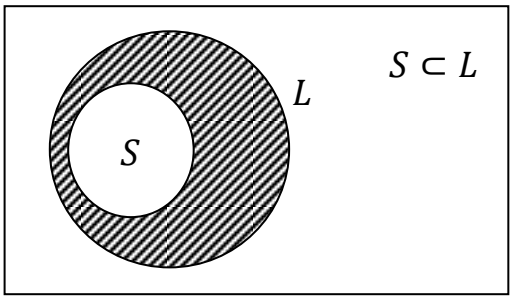

Figura 7. Diagrama de Venn, representação do conjunto S contido em L.

De outra maneira, também poderíamos representar o tempo adicional de processamento para a criação de um SNP pelo modelo 1 como:

$$A_t \propto (t_1 - t_2) \qquad (8)$$

A Figura 7, bem como a equação (8), supõe que a quantidade de $p's$ para a criação das informações é constante para qualquer x_i.

Se as informações diferem quanto à quantidade de unidades de esforço de processamento a serem utilizadas em suas respectivas criações, então a representação é um pouco mais complexa. Considere que as informações a serem criadas demandam as quantidades mostradas na Tabela 1.

Tabela 1. Demanda dos modelos por unidades de processamento.

Modelo 1		**Modelo 2**	
Informações	**p**	**Informações**	**p**
x_1	p_1	x_1	p_4
x_2	p_2	x_2	p_5
x_3	p_3		
Total	$\sum_{i=1}^{t_1} p_i$		$\sum_{i=1}^{t_2} p_i$

Novamente, se considerarmos x_1 como a informação de configuração de alelos para um loco ou SNP e que, para este último, a informação é mais facilmente criada

por ser representada por um entre dois estados, 0 e 1, então pode-se afirmar que $p_1 > p_4$. Se x_2 representa a posição do loco ou SNP o esforço computacional para suas respectivas criações é o mesmo, ou seja, $p_2 = p_5$. E finalmente, x_3 é o efeito do loco poligênico que necessita ser criado com o valor zero para simular SNP pelo modelo 1. Portanto, a quantidade de esforço p_3 só existe se o SNP for criado de acordo com o modelo 1. Dessa forma, a quantidade de tempo adicional para a simulação de um SNP pelos dois modelos é representada como:

$$A_p \propto \left(\sum_{i=1}^{t_1} p_{i1} - \sum_{i=1}^{t_2} p_{i2}\right)$$

se $\sum_{i=1}^{t_1} p_{i1} = P_1$ e $\sum_{i=1}^{t_2} p_{i2} = P_2$, então:

$$A_p \propto (P_1 - P_2) \qquad (9)$$

A quantidade de tempo adicional para a geração de um SNP pelo modelo 1 em relação ao modelo 2 é proporcional à diferença do total de unidades de esforço de processamento entre os dois modelos.

Da maneira como definido aqui, é possível que A_p em (9) seja negativo, indicando que maior tempo de processamento é demandado pelo modelo 2.

Por outro lado, uma análise do tempo de processamento associado aos modelos comparados só é completa se realizada ao nível de genoma, considerando a formação de locos poligênicos e SNPs.

Na geração dos locos e SNPs, além da criação das informações que os caracterizam, deve-se alocá-los nos cromossomos nas posições adequadas. Assim, será assumido aqui que a formação de um loco poligênico ou SNP envolve a criação das informações e a organização dessas de acordo com algum critério.

Considere um cromossomo como um conjunto (H) com informações de locos poligênicos, SNPs e locos de grande efeito:

$$H = \{h_i, i = 1, 2, \dots k\} \qquad (10)$$

em que,

$h_i =$ conjunto de elementos do cromossomo.

Se o modelo 1 for utilizado para geração de SNP, os elementos h_i são criados de acordo com (1), ou seja, conjuntos L, e cada h_i representa um conjunto L.

Se o modelo 2 for utilizado, os locos poligênicos e de grande efeito são criados de acordo com (1) e os SNPs de acordo com (2). Agora, se l locos poligênicos e s SNPs devem ser criados, H representa uma coleção de conjuntos L e S, em que os l h_i's são como em (1) e os s h_i's são como em (2). Portanto,

$$H = \{h_i, i = 1, 2, \dots l, (l+1), (l+2), \dots s\} \qquad (11)$$

De maneira simplificada, H pode ser representado como:

$$H = \{h_i, i = 1, 2\} \qquad (12)$$

em que,

$h_1 =$ conjunto de locos poligênicos de tamanho l gerados de acordo com o modelo 1, ou seja, $h_1 = \{L_i, i = 1, 2, 3, \dots l\}$; e

$h_2 =$ conjunto de SNPs de tamanho s gerados de acordo com o modelo 2, ou seja, $h_2 = \{S_i, i = 1, 2, 3, \dots s\}$.

Para a formação do genoma de um indivíduo, cada elemento de H (já criado) deve ser alocado e ordenado de acordo com sua posição no cromossomo.

Por simplicidade considere um genoma formado por um único cromossomo e, portanto, o genoma é representado pelo conjunto H. Cada elemento h_i de H possui uma posição, aleatoriamente gerada no momento de sua formação e pela qual os h_i's devem ser ordenados. Se todos os h_i's são do mesmo tipo, então existe um único conjunto de esforços de processamento (P_1) para a ordenação dos locos poligênicos e SNPs, de acordo com a posição relativa no genoma (cromossomo).

Se o modelo 2 foi adotado, então os elementos h_i não são iguais e H deve ser representado como em (12).

Nesse ponto, a ordenação das posições dos locos poligênicos e SNPs deve trabalhar com dois conjuntos, h_1 e h_2 (Figura 8).

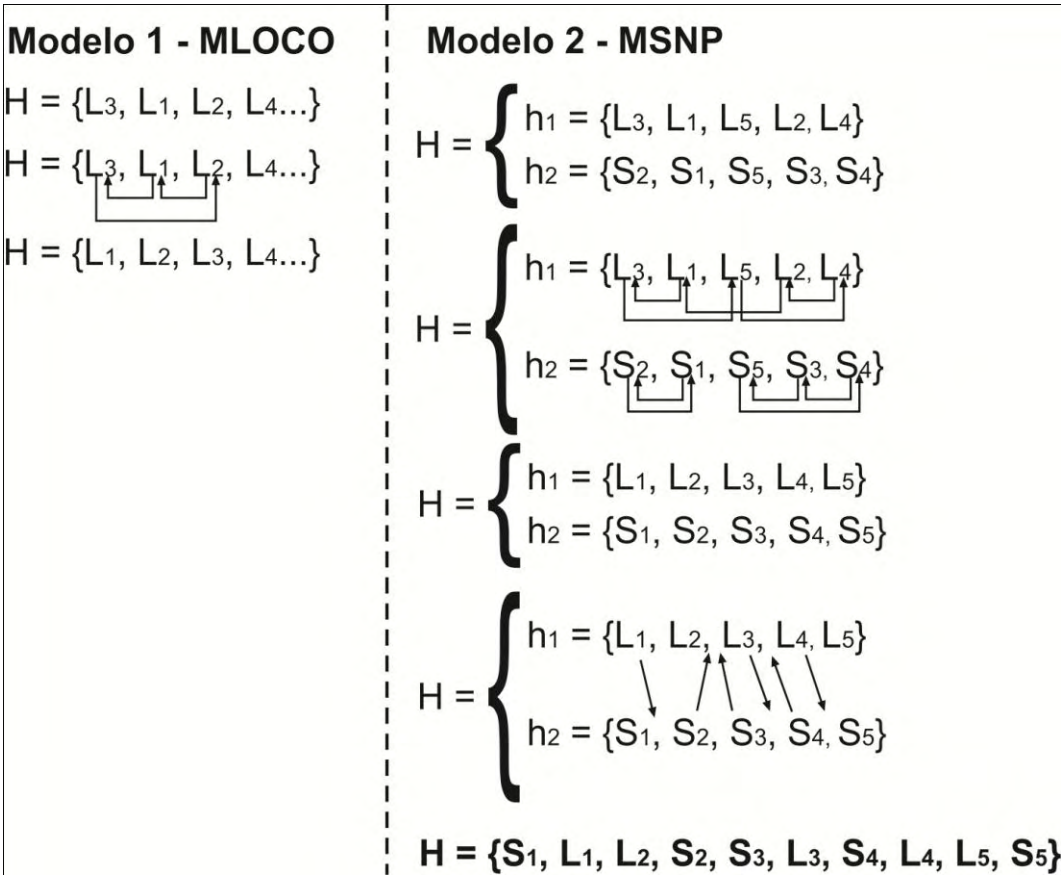

Figura 8. Ordenação dos subconjuntos de H, L e S, pela posição do loco no cromossomo.

Suponha que os conjuntos h_1 e h_2 possam ser representados como:

$$h_1 = \{L_1, L_2, \ldots L_l\};\text{ e} \qquad (13)$$

$$h_2 = \{S_1, S_2, \ldots S_l\}. \qquad (14)$$

Um conjunto de esforços de processamento (P_2) deve ser realizado para ordenar o conjunto h_1 e outro conjunto de esforços de processamento (P_3) deve ser realizado para a ordenação de h_2, com relação às suas respectivas posições no cromossomo (x_2).

Considera-se que, em média, o esforço de processamento é proporcional ao tamanho do conjunto e assim, o esforço total requerido nessa etapa é:

$$P_T = P_2 + P_3 \qquad (15)$$

Como $s+l=k$, em que k é o tamanho de H, criado de acordo com o modelo 1, pode-se dizer que $P_T=P_1$.

Entretanto, todo procedimento no processo de simulação, que demande pelo cromossomo com locos poligênicos e SNPs organizados por posição, precisa de um conjunto J, tal que:

$$J = \{h_1 \cup h_2\} \qquad (16)$$

de forma que os elementos de J estejam ordenados de acordo com x_2, ou seja, a posição no cromossomo.

A formação de J demanda a comparação dos elementos x_2 de h_1 e h_2, de forma a ordená-los de acordo com esse critério. Essa comparação entre conjuntos demanda esforços de processamento (P_4) que são, em média, proporcionais ao tamanho do maior conjunto, tipicamente, S.

Portanto, para cada criação dos elementos do genoma, a diferença em tempo de processamento pode ser representada por:

$$D_p = P_1 - P_2 \qquad (17)$$

O modelo 2, por separar os elementos em conjuntos diferentes, impõe ao programa de simulação um conjunto de esforços de processamento adicional, P_4, e, desta forma, deve utilizar mais tempo de processamento do que quando o modelo 1 é utilizado. Sendo assim:

$$A_p \propto A_{p_c} + A_{p_o} \qquad (18)$$

onde,

$$A_{p_c} \propto P_1 - P_2 \qquad (19)$$

$$A_{p_o} \propto P_1 - P_2 \propto P_1 - (P_2 + P_3 + P_4) \qquad (20)$$

como,

$$P_1 = P_2 + P_3 \qquad (21)$$

substituindo (21) em (20),

$$A_{p_o} \propto P_1 - (P_1 + P_4) \qquad (22)$$

assim,

$$A_p \propto P_1 - P_2 + P_4 \qquad (23)$$

Como P_1-P_2<<P_4 e essa diferença tende a zero, esse tempo é proporcional a P_4, o qual é proporcional ao tamanho do maior conjunto entre L e S:

$$A_p \propto P_4 \propto s$$

Assim, podemos verificar que o modelo 2 proporciona maior eficiência na utilização da memória RAM mas causa uma demanda por maior tempo de processamento. Como, de modo geral, tempo de processamento não é uma limitação séria, o modelo 2 parece ser mais adequado à simulação de SNPs.

A Tabela 2 enumera as vantagens e desvantagens dos dois modelos de acordo com os critérios acima e mais a complexidade de programação ou codificação.

Tabela 2. Comparação de eficiência entre MLOCO e MSNP.

Critérios	**Modelo 1**	**Modelo 2**
Utilização de memória RAM	*Menos eficiente*	*Mais eficiente*
Tempo de processamento	*Menor*	*Maior*
Complexidade do código	*Menor*	*Maior*
Facilidade de Manutenção	*Maior*	*Menor*

A Tabela mostra que a única característica vantajosa do modelo 2 é a economia de memória, sendo que o modelo 1 é mais vantajoso para as outras características listadas.

A economia de memória é um fator de grande peso a curto prazo, mas a longo prazo, com o barateamento da memória e aumento da capacidade dos computadores o modelo 1 torna-se competitivo por possuir características que favoreçam a manutenção do “software” e possivelmente uma maior vida útil a esse.

4 – IMPLEMENTAÇÃO

Os modelos de algoritmos MLOCO e MSNP foram implementados em liguagem de programação C++. Esta linguagem segue o paradigma de programação orientada a objetos, que é um modo natural de pensar sobre o mundo e de escrever programas de computador.

A codificação dos modelos foi incorporada a um “software” denominado LZ5.

4.1 – APRESENTAÇÃO DO SISTEMA LZ5

O LZ5, em desenvolvimento no Laboratório de Computação Cientifica Aplicada à Zootecnia (LuCCA-Z) da Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP), *Campus* de Dracena-SP, é uma ferramenta objetiva para simulação de dados e avaliação genética visando aplicações em ensino e pesquisa em genética e melhoramento animal.

Esta ferramenta é um “software-livre”, ou seja, possui código fonte aberto, disponível para a comunidade. Este pode ser modificado para fins específicos e redistribuído sem custos. Isto favorece um ambiente colaborativo, integrando centros de ensino e pesquisa interessados no desenvolvimento e aplicação do programa.

O LZ5 é constituído de 5 módulos (Figura 9).

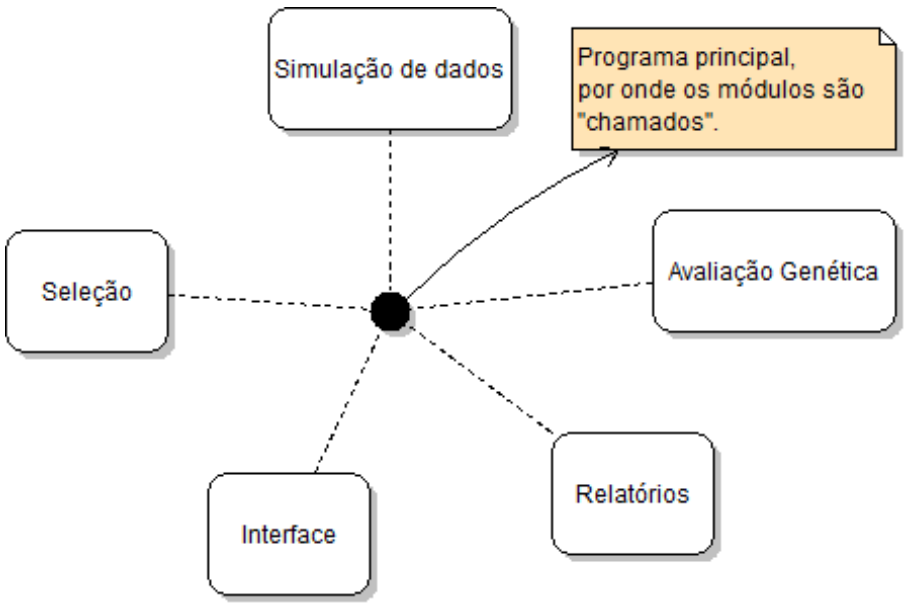

Figura 9. Representação dos módulos que compõem o simulador LZ5.

Cada um destes módulos é composto por um conjunto de classes, unidade básica que detém os atributos e as funções inerentes a criação de um objeto (por exemplo, indivíduo, cromossomo, loco). A Figura 10 apresenta o simulador LZ5, com detalhes dos módulos, utilizando a UML (“Unified Modelling Language). Esta é uma

linguagem gráfica que permite às pessoas que constroem sistemas representar seus projetos orientados a objetos usando uma notação comum (DEITEL e DEITEL, 2001).

O módulo de simulação de dados, foco para as implementações de MLOCO e MSNP, é constituído pelas classes “Parameters”, “Population”, “Individual”, “Genome”, “Chromosome” e “Locus”. O módulo de Seleção é constituído pelas classes “Sel”, “RanSel” e “IndSel”. As duas últimas derivam da classe Sel. O módulo Interface detém as classes “lz5_Result_Struct”, “Lz5_Interpreter”, “Lz5_Comands_db” e “Prompt”. Por fim, os módulo de Avaliação Genética e Relatórios estão em desenvolvimento.

As relações existentes entre as classes, representadas pelas setas na Figura 12, seguem o seguinte padrão:

- - - - - - → “Dependência”;

——◆ “Composição”; e

——▷ “Herança”.

A relação de “Dependência” liga um objeto a outro, se um objeto não for criado o outro também não existirá. Na relação de “Composição” uma classe integra a outra de forma a se comporem. Por último a relação de “Herança”, a qual uma classe é fixada como base e possui informações que podem ser compartilhadas com as classes chamadas derivadas, que necessitam das mesmas informações da classe base para serem instanciadas, além de suas funções peculiares.

Essa visão geral do sistema por meio da UML facilita a visualização das diferenças nas implementações dos algoritmos propostos, realizadas no módulo de simulação de dados.

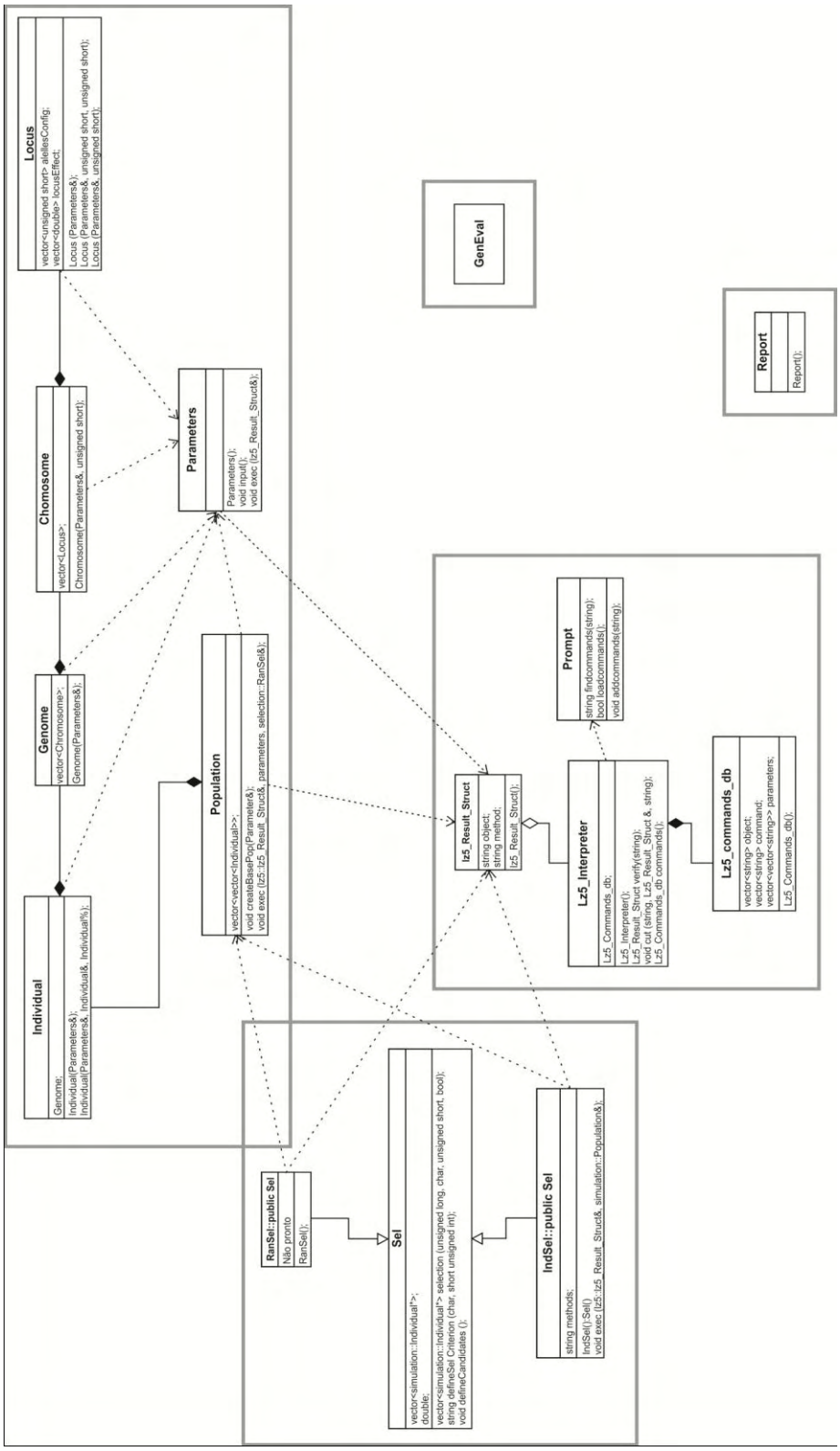

Figura 10. Diagrama de classes da UML, representando os possíveis objetos de simulação do “software” LZ5 dentro de cada módulo.

4.2 – MODELO MLOCO

A Figura 11 representa uma simplificação do código C++ implementado para este modelo.

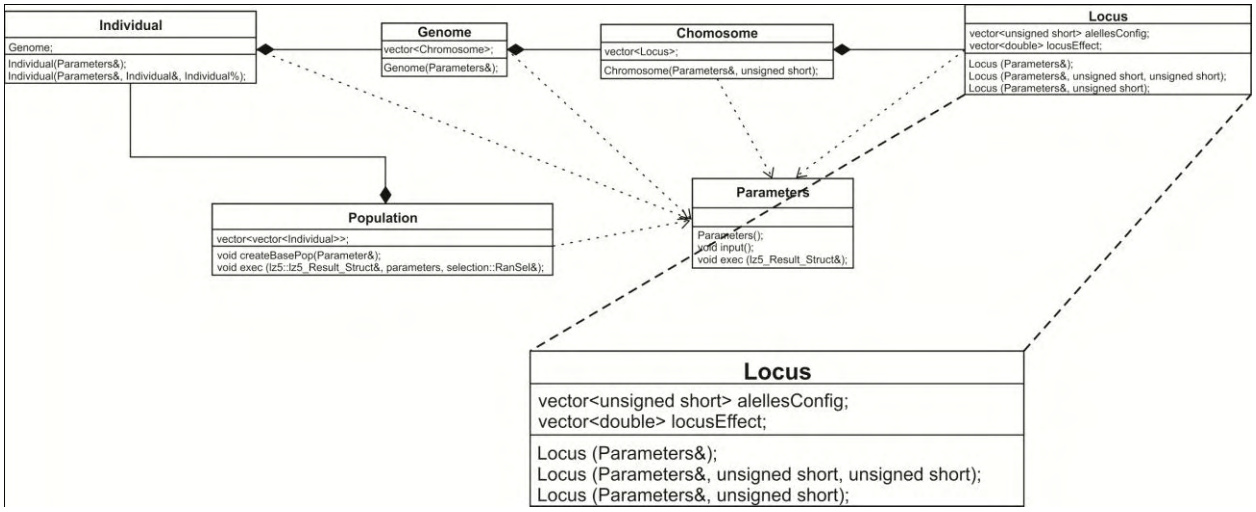

Figura 11. Diagrama de classes de acordo com a UML. Módulo de simulação de dados. A classe **Locus** possui 3 construtores específicos, um para cada tipo de loco a ser criado (SNP, QTL e Poligene).

A classe **Locus** representa o conjunto L descrito em (2) na seção 2.1.1. Esta classe pode criar três tipos de loco (poligene, SNP e QTL). Mesmo não possuindo efeito, os SNPs terão esta variável em seu conjunto, armazenando o valor 0.

Esta única classe compõe a classe **Chromosome** que é representada pelo conjunto H, descrito em (10) na seção 3.2.

Considerando-se um único cromossomo, por exemplo, os objetos **Locus** estão armazenados em posições contíguas de memória (único vetor), facilitando a busca e ordenação dos locos pela posição no genoma, variável contida em cada conjunto L.

4.3 – MODELO MSNP

A Figura 12 representa uma simplificação do código C++ implementado para este modelo.

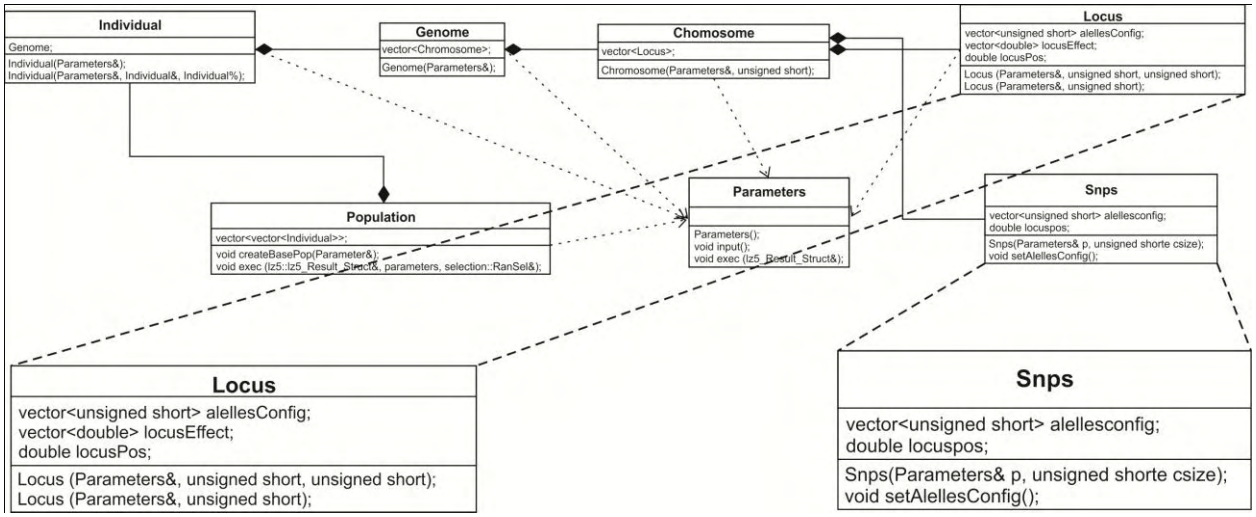

Figura 12. Diagrama de classes da UML. Módulo de simulação de dados. A classe **Snps**, juntamente com a classe **Locus**, compõem a classe **Chromosome**.

A classe **Snps** representa o conjunto S descrito em (3) na seção 2.1.2. Esta classe cria um loco marcador tipo SNP, apenas com o conjunto de informações necessárias. A classe **Locus** em MSNP representa o conjunto L, como no algoritmo MLOCO.

a classe **Chromosome** é agora composta pelas classes **Snps** e **Locus**.

Para este modelo, o conjunto H descrito em (11) e (12) da seção 3.2 representa a classe **Chromosome**. No entanto, os SNPs são armazenados na memória em posições não contíguas às de locos poligênicos e QTLs. Este fato gera uma complexidade para busca e ordenação da posição dos locos no genoma em processos, demandando maior tempo de processamento.

5 – SIMULAÇÃO

Foi simulada uma população-base (pb) hipotética, constituida por 350 indivíduos, 50% machos e 50% fêmas. A expressão de machos e fêmeas em porcentagem é devido a amostragem de números aleatórios (y), entre 0 e 1, para a identificação do sexo dos indivíduos. Se y < 0.5, o indivíduo é classificado como fêmea, se y ≥ 0.5, o

indivíduo é classificado como macho. O genoma de cada um dos indivíduos desta população é composto por um único cromossomo de tamanho 1000cM.

Simulações com 5, 10 e 20 gerações sobrepostas foram realizadas. A cada geração 150 acasalamentos ao acaso ocorreram, produzindo 150 novos indivíduos. Assim, para 5 gerações o número total de indivíduos foi de (pb+750), para 10 foi (pb+1500) e para 20 foi (pb+3000).

Foi testado o desempenho computacional (consumo de memória e processamento) para cada uma das simulações de gerações (5, 10 e 20) simulando 1.000, 3.000, 5.000 e 10.000 SNPs por indivíduo, com mais 500 locos poligenicos (Tabela 3).

Tabela 3. Representação das simulações executadas.

	Quantidade de SNPs/Indivíduo											
	1000			3000			5000			10000		
nº de Gerações	5	10	20	5	10	20	5	10	20	5	10	20
nº de Indivíduos	1100	1850	3350	1100	1850	3350	1100	1850	3350	1100	1850	3350

O código desenvolvido para ambos os modelos testados por meio de simulação estão nos apêndices.

5.1 – RESULTADOS

5.1.1 – MLOCO

Cada SNP simulado por este modelo, ou seja, o conjunto L, definido em (1) na seção 2.1, consumiu 72 bytes de memória RAM.

Para as simulações apresentadas na Tabela 3, os resultados de consumo de memória devido a criação de SNPs, para toda a população, nos diferentes cenários, estão apresentados na Tabela 4.

Tabela 4. Consumo de memória RAM pelo total de SNPs simulados na população para as diferentes simulações executadas (MLOCO).

	Consumo de memória por objetos SNPs (Mb)			
	SNPs			
Gerações	**1000**	**3000**	**5000**	**10000**
5	75,5310	226,5930	377,6550	755,3101
10	127,0294	381,0883	635,1471	1270,2942
20	230,0262	690,0787	1150,1312	2300,2625

Os tempos de processamento das simulações (Tabela 3) estão na Tabela 5.

Tabela 5. Tempo de processamento (execução do programa LZ5) utilizado para as diferentes simulações relizadas (MLOCO).

	Tempo de processamento (segundos)			
	SNPs			
Gerações	**1000**	**3000**	**5000**	**10000**
5	7,54	12,06	20,76	44,09
10	7,5	21,24	36,71	78,94
20	13,8	39	67,17	143,27

5.1.2 – MSNP

Cada SNP simulado por este modelo, ou seja, o conjunto S, definido em (3) na seção 2.2, consumiu 40 bytes de memória RAM.

Para a estrutura de simulação apresentada na Tabela 3, os resultados de consumo de memória devido a criação de SNPs, para toda a população, nos diferentes cenários, estão apresentados na Tabela 5.

Tabela 6. Consumo de memória RAM pelo total de SNPs simulados na população para as diferentes simulações executadas (MSNP).

	Consumo de memória por objetos SNPs (Mb)			
	SNPs			
Gerações	**1000**	**3000**	**5000**	**10000**
5	41,9617	125,8850	209,8083	419,6167
10	70,5719	211,7157	352,8595	705,7190
20	127,7924	383,3771	638,9618	1277,9236

Os tempos de processamento das simulações (Tabela 3) estão na Tabela 7.

Tabela 7. Tempo de processamento (execução do programa LZ5) utilizado para as diferentes simulações relizadas (MLOCO).

	Tempo de processamento (segundos)			
	SNPs			
Gerações	**1000**	**3000**	**5000**	**10000**
5	4,7	16,38	27,37	59,09
10	8,39	27,5	46,75	84,42
20	16,15	45,18	80,34	165,7

5.1.3 – CONSUMO DE MEMÓRIA MLOCO x MSNP

A eficiência relativa quanto à utilização de memória, como definido em (4) e (6) na seção 3.1, para a simulação de (pb+3000) indivíduos com 10.000 SNPs cada e 20 gerações é:

$$E_m = \frac{1277{,}9236}{2300{,}2625} = 0{,}55$$

isso significa que MSNP consome 55% menos memória comparado ao MLOCO, o que lhe confere maior eficiência, como apresentado na Tabela 2.

5.1.4 – TEMPO DE PROCESSAMENTO MLOCO x MSNP

A demanda de processamento entre os dois modelos foi comparada pelo tempo de processamento adicional, representado pela Figura 7 e por (8) na seção 3.2.

Considerando a mesma simulação do ítem 5.1.3, a diferença em tempo de processamento, definida em (17) é:

$$D_t = 143{,}27 - 165{,}7 = -22{,}43s$$

O valor negativo indica que o modelo 1 é mais eficiente em tempo de processamento comparado ao modelo 2. Este fato reside na complexidade de ordenação dos locos para MSNP. Neste algoritmo, os diferentes tipos de locos (SNPs e poligenes) estão armazenados em vetores específicos para seu tipo. Isto implica em maior número de passos durante a amostragem de alelos para formação de um novo indivíduo. Essa amostragem é feita pelo tipo de loco e sua posição. Estando em vetores diferentes, maior é a demanda por unidades de processamento.

6 - CONCLUSÕES

Em estudos de simulação, que envolvam grande quantidade de informações, o modelo SNP é a alternativa apropriada para realização de simulações de dados SNPs. Por outro lado, o modelo loco é uma alternativa viável se não houver limitação por consumo de memória, pois a implementação e manutenção do sistema que utilizar este modelo será mais simples, além de proporcionar maior velocidade de processamento.

REFERÊNCIAS

AULCHENKO, Y. S.; RIPKE, S.; ISAACS, A.; VAN DUIJN, C. M. GenABEL: an R library for genome-wide association analysis. **Bioinformatics**. p.1294-1296. 2007.

BIOCONDUCTOR. Disponível em: <http://www.bioconductor.org>. Acesso em: 08 abr. 2010.

CAETANO, A. R. Marcadores SNPs: Conceitos básicos, aplicações no manejo e no melhoramento animal e perspectivas para o futuro. **Rev. Bras. de Zootec.**, v.38, p.64-71. 2009.

ECK, S. H.; BENET-PAGÈS, A.; KRZYSZTOF, F.; MEITINGER, T.; FRIES, R.; STROM, T. M. Whole genome sequencing of a single *Bos taurus* animal for single nucleotide polymorphism discovery. **Genome Biology.** 2009.

EUCLYDES, R. F. Uso do sistema para simulação *Genesys* na avaliação de métodos de seleção clássicos e associados a marcadores moleculares. Viçosa:UFV, 1996.

FALCONER, D. S.; MACKAY, T. F. C. Introduction to Quantitative Genetics., ed.4 Longman, 1996.

FISCHER, R. A. The correlation between relatives: the supposition of mendelian inheritance. **Transactions of the royal society of Edinburgh.**, v.59, p.399. 1918.

HACIA, J. G. Resequencing and mutational analysis using oligonucleotide microarrays. **Nature Genetics Supplement.**, v.21, p.42-47, 1999.

HAYES, B. J.; BOWMAN, P. J.; CHAMBERLAIN, A. J.; GODDARD, M. E. Genomic selection in dairy cattle: Progress and challenges. **J. Dairy Sci.**, v.92, p.433-443. 2009.

HAYES, B. J.; GODDARD, M. E. The distribution of the effects of genes affecting quantitative traits in livestock. **Genet. Sel. Evol.**, v.33, ed.3, p.209-229. 2001.

HENDERSON, C. R. A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. **Biometrics**, v.32, p.69-83. 1976.

HOGGART, C. J.; CHADEAU-HYAM, M.; CLARK, T. G.; LAMPARIELLO, R.; WHITTAKER, J. C.; DE IORIO, M.; BALDING, D. J. Sequence-Level Population Simulations Over Large Genomic Regions. **Genetics.**, v.177, p.1725-1731. 2007.

Illumina. BovineSNP50 genotyping BeadChip. **Illumina, Inc.**, 370-2007-029. p.1-4. 2010. Disponível em: http://www.illumina.com/Documents/products/datasheets/datasheet_bovine_snp50.pdf. Acessado em: 08/2010.

Illumina. BovineHD genotyping BeadChip. **Illumina, Inc.**, 370-2010-018, p.1-4. 2010. Disponível em: http://www.illumina.com/Documents/products/datasheets/datasheet_bovineHD.pdf. Acessado em: 08/2010.

INTERNATIONAL SHEEP GENOMICS CONSORTIUM. A genome wide survey of SNP variation reveals the genetic structure of sheep breeds. **PLOS ONE**, v.4, 13p. 2009.

KIM, S.; MISRA, A. SNP genotyping: Technologies and biomedical applications. **Annual Review of Biomedical Engineering**. v.9, p.289-320. 2007.

MEUWISSEN, T. H. E.; HAYES, B. J.; GODDARD, M. E. Prediction of total genetic value using genome-wide dense marker maps. **Genetics**, v.157, p.1819-1829. 2001.

SANGER, F.; NICKLEN, S.; COULSON, A. R. DNA sequencing with chain-terminating inhibitors. Proceedings of the National Academy of Sciences., v.74, p.5463-5467. 1977.

THE BOVINE HAPMAP CONSORTIUM. Genome-Wide survey of SNP variation uncovers the genetic structure of cattle breeds. **Science.,** v.324, p.528-532.

THE R PROJECT FOR STATISTICAL COMPUTING. Disponível em:<http://www.r-project.org/>. Acesso em: 10 nov. 2009.

VAN TASSEL C. P.; SMITH, T. P.; MATUKUMALLI, L. K.; TAYLOR, J. F.; SCHNABEL, R. D.; LAWLEY, C. T.; HAUDENSCHILD, C. D.; MOORE, S. S.; WARREN, W. C.; SONSTEGARD, T. S. SNP discovery and allele frequency estimation by deep sequencing of reduced representation libraries. **Nature Methods.**, v.5, ed.3, p.247-252. 2008.

VIGNAL, A.; MILAN, D.; SAN CRISTOBAL, M.; EGGEN, A. A review on SNP and other types of molecular markers and their use in animal genetics. **Genet. Sel. Evol.**, v.34, p.275-305. 2002.

APÊNDICE A – locus.h: código fonte em C++ contendo os protótipos das funções necessárias para simular locos SNPs pelo MLOCO.

```
#ifndef _LOCUS_H_
#define _LOCUS_H_

-----------------------------------------------------------------

#include "parameters.h"
#include <iostream>
using std::vector;

-----------------------------------------------------------------

namespace simulation
{
  class Locus
  {
  private:
    vector<unsigned short> alellesConfig;
    vector<double> locusEffect;
    unsigned short traitId;
    double locusPos;
    char type;

-----------------------------------------------------------------

  public:
    inline
Locus():alellesConfig(),locusEffect(),traitId(),locusPos(0),type
('m'){};
    Locus(Parameters& p);
    Locus(Parameters& p,unsigned short tId,unsigned short qId);
    Locus(Parameters& p, unsigned short csize);

    inline void setLocusEffect(unsigned short i,double
e){locusEffect[i]=e;}
    void setLocusEffect(Parameters& p);
    inline void setAlellesConfig(unsigned short i,unsigned short
j,unsigned short k,unsigned short
l){alellesConfig[i]=k;alellesConfig[j]=l;}
    void setAlellesConfig(Parameters& p);
    void setAlellesConfig(Parameters& p,unsigned short
a,unsigned short qId);
    void setAlellesConfig();
    void setQtlLocusEffect(Parameters& p,unsigned short
i,unsigned short j);
    inline void setTraitId(unsigned short t){traitId=t;}
    void setTraitId(Parameters& p,unsigned lc);
    inline void setLocusPos(double p){locusPos=p;}
```

```
    inline void setType(char c){type=c;}
    inline unsigned short getAlellesConfig(unsigned short
i)const{return alellesConfig[i];}
    inline double getLocusEffect(unsigned short i)const{return
locusEffect[i];}
    inline unsigned short getTraitId()const{return traitId;}
    inline double getLocusPos()const{return locusPos;}
    inline char getType()const{return type;}
  };
}

#endif
-----------------------------------------------------------------
-----------------------------------------------------------------
```

APÊNDICE B – locus.cpp: código fonte em C++ contendo a definição das funções necessárias para simular locos SNPs pelo MLOCO.

```
#include"locus.h"
#include<gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include<ctime>

-----------------------------------------------------------------

extern gsl_rng* prand;
namespace simulation{
  static unsigned pLociCounter=0;

-----------------------------------------------------------------

  Locus::Locus(Parameters&
p):alellesConfig(2,0),locusEffect(p.getNumTraits(),0),traitId(0)
,locusPos(0),type('p'){
    ++pLociCounter;

    setTraitId(p,pLociCounter);
    setAlellesConfig(p);
    setLocusEffect(p);

    vector<pair<unsigned short> >::iterator
pit=p.getNumPol().end()-1;
    if(pLociCounter==p.getNumPol(pit-
>a)||pLociCounter>p.getNumPol(pit->a))
    pLociCounter=0;
  }

-----------------------------------------------------------------

  Locus::Locus(Parameters& p,unsigned short tId,unsigned short
qId):alellesConfig(2,0),locusEffect(p.getNumTraits(),0),traitId(
tId),locusPos(0),type('q'){
    setAlellesConfig(p,tId,qId);
    setQtlLocusEffect(p,tId,qId);
  }

-----------------------------------------------------------------

  Locus::Locus(Parameters& p,unsigned short
csize):alellesConfig(2,0),locusEffect(p.getNumTraits(),0),traitI
d(),locusPos(0),type('m'){
    setAlellesConfig();

  }
-----------------------------------------------------------------
```

```
  void Locus::setTraitId(Parameters& p,unsigned lc){
    using std::sort;

    if(p.getNumTraits()==1)
      traitId=0;
    else{

sort(p.getNumPol().begin(),p.getNumPol().end(),sortNumPol());

      vector<pair<unsigned short> >::iterator vit;

for(vit=p.getNumPol().begin();vit!=p.getNumPol().end();++vit){
     if(lc<vit->b||lc==vit->b){
       traitId=vit->a;
       break;
     }
      }
    }

  }
```

```
  void Locus::setAlellesConfig(Parameters& p){

    double rNum;

    //*******************************************************
    //paternal alelle sampling-BEGINNING                    *
    //*******************************************************
    rNum=gsl_rng_uniform(prand);
    unsigned short pAlelle=0;

    if (rNum>p.getFreqAlelle(traitId))
      {
     pAlelle = 1;
      }
    //*******************************************************
    //paternal alelle sampling-END                          *
    //*******************************************************


    //*******************************************************
    //maternal alelle sampling-BEGINNING                    *
    //*******************************************************
    rNum=gsl_rng_uniform(prand);
    unsigned short mAlelle=0;
```

```
    if (rNum>p.getFreqAlelle(traitId))
      {
     mAlelle = 1;
      }

  //*********************************************************
    //maternal alelle sampling-END                           *
    //*********************************************************

    alellesConfig[0] = pAlelle;
    alellesConfig[1] = mAlelle;
  }

-----------------------------------------------------------------

  void Locus::setAlellesConfig(Parameters& p,unsigned short
tId,unsigned short qId){

    double rNum;

    //*********************************************************
    //paternal alelle sampling-BEGINNING                      *
    //*********************************************************
    rNum=gsl_rng_uniform(prand);
    unsigned short pAlelle=0;

    if (rNum>p.getQtlsAlellesFreq(tId,qId))
      {
     pAlelle = 1;
      }
    //*********************************************************
    //paternal alelle sampling-END                            *
    //*********************************************************


    //*********************************************************
    //maternal alelle sampling-BEGINNING                      *
    //*********************************************************
    rNum=gsl_rng_uniform(prand);
    unsigned short mAlelle=0;

    if (rNum>p.getQtlsAlellesFreq(tId,qId))
      {
     mAlelle = 1;
      }
```

```
    //*********************************************************
    //maternal alelle sampling-END                            *
    //*********************************************************

    alellesConfig[0] = pAlelle;
    alellesConfig[1] = mAlelle;
  }
```

```
 void Locus::setAlellesConfig(){

    double rNum;

    //*********************************************************
    //paternal alelle sampling-BEGINNING                      *
    //*********************************************************
    rNum=gsl_rng_uniform(prand);
    unsigned short pAlelle=0;

    if (rNum>0.5)
      {
     pAlelle = 1;
      }
    //*********************************************************
    //paternal alelle sampling-END                            *
    //*********************************************************


    //*********************************************************
    //maternal alelle sampling-BEGINNING                      *
    //*********************************************************
    rNum=gsl_rng_uniform(prand);
    unsigned short mAlelle=0;

    if (rNum>0.5)
      {
     mAlelle = 1;
      }
    //*********************************************************
    //maternal alelle sampling-END                            *
    //*********************************************************

    alellesConfig[0] = pAlelle;
    alellesConfig[1] = mAlelle;
  }
```

```
  void Locus::setLocusEffect(Parameters& p){
    for (unsigned short i=traitId;i<p.getNumTraits();++i){
      if(alellesConfig[0]==0 && alellesConfig[1]==0)
     locusEffect[i]=p.getPolVal(traitId,i);
      else if(alellesConfig[0]==1 && alellesConfig[1]==1)
     locusEffect[i]=-(p.getPolVal(traitId,i));
    }
  }

-----------------------------------------------------------------

  void Locus::setQtlLocusEffect(Parameters& p,unsigned short
i,unsigned short j){
    if(alellesConfig[0]==0 && alellesConfig[1]==0)
      locusEffect[i]=p.getQtlsGenEff(i,j);
    else if(alellesConfig[0]==1 && alellesConfig[1]==1)
      locusEffect[i]=-p.getQtlsGenEff(i,j);
  }

}
-----------------------------------------------------------------
-----------------------------------------------------------------
```

APÊNDICE C – individual.h: código fonte em C++ contendo o protótipo da função necessária para ordenar os locos nos cromossomos de acordo com sua posição (MLOCO).

```
#ifndef _INDIVIDUAL_H_
#define _INDIVIDUAL_H_

#include "genome.h"

-----------------------------------------------------------------

namespace simulation
{
  class Individual
  {
  private:
    Genome genome;
    vector<double> phenVal;
    vector<double> adGenVal;
    vector<double> envDevVal;
    unsigned sireId;
    unsigned damId;
    unsigned indId;
    char gender;

-----------------------------------------------------------------

  public:
    inline
Individual():genome(),phenVal(),adGenVal(),envDevVal(),sireId(0)
,damId(0),indId(0),gender('m'){};
    Individual(Parameters& p);
    Individual(Parameters& p,Individual& c,Individual& d);

    inline void setPhenVal(unsigned short i,double p)
{phenVal[i]=p;}
    inline void setAdGenVal(unsigned short i, double a)
{adGenVal[i]=a;}
    inline void setEnvDevVal(unsigned short i, double
e){envDevVal[i]=e;}
    inline void setSireId(unsigned s){sireId=s;}
    inline void setDamId(unsigned d){damId=d;}
    inline void setIndId(unsigned i){indId=i;}
    inline void setGender(char g){gender=g;}

    inline double getPhenVal(unsigned short i)const{return
phenVal[i];}
    inline double getAdGenVal(unsigned short i)const {return
adGenVal[i];}
    inline double getEnvDevVal(unsigned short i)const{return
envDevVal[i];}
```

```
    inline unsigned getSireId()const{return sireId;}
    inline unsigned getDamId()const{return damId;}
    inline unsigned getIndId()const{return indId;}
    inline char getGender()const{return gender;}

    inline Genome& getGenome(){return genome;}

    void calcAdGenVal(Parameters& p);
    void calcPhenVal(Parameters& p);

    char pickGender();
    void zygogenesis(Parameters& p,Individual& s,Individual& d);
  };
}
#endif
-----------------------------------------------------------------
-----------------------------------------------------------------
```

APÊNDICE D – individual.cpp: código fonte em C++ contendo as definições das funções necessárias para ordenar os locos nos cromossomos de acordo com sua posição (MLOCO).

```
#include"individual.h"
#include<gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include<ctime>
#include<cmath>

#include<iostream>

-----------------------------------------------------------------

using namespace std;
extern gsl_rng* prand;
namespace simulation{

  static unsigned long indCounter=0;

  Individual::Individual(Parameters&
p):genome(p),phenVal(p.getNumTraits()),adGenVal(p.getNumTraits()
),envDevVal(p.getNumTraits()),sireId(0),damId(0),indId(indCounte
r++),gender(pickGender()){
    calcAdGenVal(p);
    calcPhenVal(p);
  }

-----------------------------------------------------------------

  Individual::Individual(Parameters& p,Individual& s,Individual&
d):genome(p),phenVal(p.getNumTraits()),adGenVal(p.getNumTraits()
),envDevVal(p.getNumTraits()),sireId(s.getIndId()),damId(d.getIn
dId()),indId(indCounter++),gender(pickGender()){
    zygogenesis(p,s,d);
    calcAdGenVal(p);
    calcPhenVal(p);
  }

-----------------------------------------------------------------

  void Individual::calcAdGenVal(Parameters& p){
    vector<Chromosome>::iterator cit;
    vector<Locus>::iterator lit;


for(cit=genome.getGenom().begin();cit!=genome.getGenom().end();+
+cit){
      for(lit=cit->getChrom().begin();lit!=cit-
>getChrom().end();++lit){
     for(unsigned short i=0;i<p.getNumTraits();++i){
```

```
        adGenVal[i]+=lit->getLocusEffect(i);
      }
       }
     }
   }
```

```
  void Individual::calcPhenVal(Parameters& p){
    for(unsigned short i=0;i<p.getNumTraits();++i){
      envDevVal[i] = gsl_ran_gaussian(prand,
sqrt(p.getGenAdCov(i,i)*(1/p.getHeritability(i)-1)));
      phenVal[i]=p.getTraitMeans(i)+adGenVal[i]+envDevVal[i];
    }
  }
```

```
  char Individual::pickGender(){
    double rNumber=gsl_rng_uniform(prand);
    if(rNumber<0.5)
      return 'f';
    else
      return 'm';
  }
```

```
  void Individual::zygogenesis(Parameters& p,Individual&
s,Individual& d){

//*************************************************************
//*Sampling a chromosome in the individual's parents-BEGINNING *
//*************************************************************
    unsigned short sIndex=0;
    unsigned short dIndex=0;

    double sNumber=gsl_rng_uniform(prand);
    if(sNumber<0.5)
      sIndex=1;

    double dNumber=gsl_rng_uniform(prand);
    if(dNumber<0.5)
      dIndex=1;
```

```
//*********************************************************
//*Sampling a chromosome in the individual's parents-END *
//*********************************************************
    vector<Chromosome>::iterator icit;
    vector<Chromosome>::iterator scit;
    vector<Chromosome>::iterator dcit;
    vector<Locus>::iterator ilit;
    vector<Locus>::iterator slit;
    vector<Locus>::iterator dlit;

    double rbNumber=0;


for(icit=genome.getGenom().begin(),scit=s.genome.getGenom().begi
n(),dcit=d.genome.getGenom().begin();icit!=genome.getGenom().end
(),scit!=s.genome.getGenom().end(),dcit!=d.genome.getGenom().end
();++icit,++scit,++dcit){
      for(ilit=icit->getChrom().begin(),slit=scit-
>getChrom().begin(),dlit=dcit->getChrom().begin();ilit!=icit-
>getChrom().end(),slit!=scit->getChrom().end(),dlit!=dcit-
>getChrom().end();++ilit,++slit,++dlit){
//*************************************************************
//Test and execute recombination for sire's chromosomes-
BEGINNING *
//*************************************************************
     rbNumber=gsl_rng_uniform(prand);
     if(rbNumber<p.getRecombRate()){
       if(sIndex==0)
         sIndex=1;
       else
         sIndex=0;
     }
//*********************************************************
//*Test and execute recombination for sire's chromosomes-END *
//*********************************************************


//*************************************************************
//*Test and execute recombination for dam's chromosomes-
BEGINNING *
//*************************************************************
     rbNumber=gsl_rng_uniform(prand);
     if(rbNumber<p.getRecombRate()){
       if(dIndex==0)
         dIndex=1;
       else
         dIndex=0;
```

```
      }
//************************************************************
//*Test and execute recombination for dam's chromosomes-END *
//************************************************************

      if(slit->getType()=='p')
        ilit->setType('p');
      else if(slit->getType()=='q')
        ilit->setType('q');
      else
        ilit->setType('m');

      ilit->setLocusPos(slit->getLocusPos());
      ilit->setAlellesConfig(0,1,slit-
>getAlellesConfig(sIndex),dlit->getAlellesConfig(dIndex));
      ilit->setTraitId(slit->getTraitId());
      if(ilit->getType()=='p')
        ilit->setLocusEffect(p);
      else if(ilit->getType()=='q')
        ilit->setQtlLocusEffect(p,ilit->getTraitId(),(icit-
genome.getGenom().begin()));
      else
        ilit->setLocusEffect(ilit->getTraitId(),0);
       }
     }
   }
}
```

APÊNDICE E – snp.h: código fonte em C++ contendo os protótipos das funções necessárias para simular um loco SNP pelo MSNP.

```
#ifndef _SNP_H_
#define _SNP_H_

-----------------------------------------------------------------

#include "locus.h"
#include "parameters.h"
#include <iostream>
using std::vector;

-----------------------------------------------------------------

namespace simulation
{
  class Snps : public Locus{
  private:
    vector<unsigned short> alellesConfig;
    unsigned short traitId;
    double locusPos;
    char type;

-----------------------------------------------------------------

  public:
    Snps(Parameters& p, unsigned short csize);
    void setAlellesConfig();
  };
}
#endif
-----------------------------------------------------------------
-----------------------------------------------------------------
```

APÊNDICE F – snp.cpp: código fonte em C++ contendo as definições das funções inerentes a simulação de SNPs pelo MSNP.

```
#include"snp.h"
#include<gsl/gsl_rng.h>
#include<ctime>

-----------------------------------------------------------------

extern gsl_rng* prand;

namespace simulation{

  static unsigned long mLociCounter=0;
-----------------------------------------------------------------

  Snps::Snps(Parameters& p,unsigned short
csize):Locus(),alellesConfig(2,0),traitId(),locusPos(0),type('m'
){
    setAlellesConfig();
    locusPos=double(mLociCounter*p.getMarkerDist());
    ++mLociCounter;
    if(mLociCounter>csize)
      mLociCounter=0;
  }

-----------------------------------------------------------------

  void Snps::setAlellesConfig(){

    double rNum;

    //*******************************************************
    //paternal alelle sampling-BEGINNING                    *
    //*******************************************************
    rNum=gsl_rng_uniform(prand);
    unsigned short pAlelle=0;

    if (rNum<0.5)
      {
     pAlelle = 1;
      }
    //*******************************************************
    //paternal alelle sampling-END                          *
    //*******************************************************


    //*******************************************************
    //maternal alelle sampling-BEGINNING                    *
    //*******************************************************
```

```
    rNum=gsl_rng_uniform(prand);
    unsigned short mAlelle=0;

    if (rNum<0.5)
      {
     mAlelle = 1;
      }
    //*********************************************************
    //maternal alelle sampling-END                            *
    //*********************************************************

    alellesConfig[0] = pAlelle;
    alellesConfig[1] = mAlelle;
  }
}
-----------------------------------------------------------------
-----------------------------------------------------------------
```

APÊNDICE G – individual.h: código fonte em C++ contendo o protótipo da função necessária para ordenar os locos nos cromossomos de acordo com sua posição (MSNP).

```
#ifndef _INDIVIDUAL_H_
#define _INDIVIDUAL_H_

#include "genome.h"

-----------------------------------------------------------------
namespace simulation
{

  class Individual
  {
  private:
    Genome genome;
    vector<double> phenVal;          // vector of phenotypic
values;
    vector<double> adGenVal;         // vector of additive
genetic value;
    vector<double> envDevVal;        // vector of environment
deviate;
    unsigned sireId;                 // sire id;
    unsigned damId;                  // dam id;
    unsigned indId;                  // individual id;
    char gender;                     // individual gender.

-----------------------------------------------------------------

  public:
    inline
Individual():genome(),phenVal(),adGenVal(),envDevVal(),sireId(0)
,damId(0),indId(0),gender('m'){};
    Individual(Parameters& p);
    Individual(Parameters& p,Individual& c,Individual& d);

    inline void setPhenVal(unsigned short i,double p)
{phenVal[i]=p;}
    inline void setAdGenVal(unsigned short i, double a)
{adGenVal[i]=a;}
    inline void setEnvDevVal(unsigned short i, double
e){envDevVal[i]=e;}
    inline void setSireId(unsigned s){sireId=s;}
    inline void setDamId(unsigned d){damId=d;}
    inline void setIndId(unsigned i){indId=i;}
    inline void setGender(char g){gender=g;}

    inline double getPhenVal(unsigned short i)const{return
phenVal[i];}
```

```
    inline double getAdGenVal(unsigned short i)const {return
adGenVal[i];}
    inline double getEnvDevVal(unsigned short i)const{return
envDevVal[i];}
    inline unsigned getSireId()const{return sireId;}
    inline unsigned getDamId()const{return damId;}
    inline unsigned getIndId()const{return indId;}
    inline char getGender()const{return gender;}

    inline Genome& getGenome(){return genome;}

    void calcAdGenVal(Parameters& p);
    void calcPhenVal(Parameters& p);

    char pickGender();
    void zygogenesis(Parameters& p,Individual& s,Individual& d);
  };
}
#endif
-----------------------------------------------------------------
-----------------------------------------------------------------
```

APÊNDICE H – individual.cpp: código fonte em C++ contendo as definições das funções necessárias para ordenar os locos nos cromossomos de acordo com sua posição (MSNP).

```
#include"individual.h"
#include<gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include<ctime>
#include<cmath>

-----------------------------------------------------------------

extern gsl_rng* prand;
namespace simulation{

  static unsigned long indCounter=0;

  Individual::Individual(Parameters&
p):genome(p),phenVal(p.getNumTraits()),adGenVal(p.getNumTraits()
),envDevVal(p.getNumTraits()),sireId(0),damId(0),indId(indCounte
r++),gender(pickGender()){
    calcAdGenVal(p);
    calcPhenVal(p);
  }

-----------------------------------------------------------------

  Individual::Individual(Parameters& p,Individual& s,Individual&
d):genome(p),phenVal(p.getNumTraits()),adGenVal(p.getNumTraits()
),envDevVal(p.getNumTraits()),sireId(s.getIndId()),damId(d.getIn
dId()),indId(indCounter++),gender(pickGender()){
    zygogenesis(p,s,d);
    calcAdGenVal(p);
    calcPhenVal(p);
  }

-----------------------------------------------------------------

  void Individual::calcAdGenVal(Parameters& p){
    vector<Chromosome>::iterator cit;
    vector<Locus>::iterator lit;

for(cit=genome.getGenom().begin();cit!=genome.getGenom().end();+
+cit){
      for(lit=cit->getChrom().begin();lit!=cit-
>getChrom().end();++lit){
     for(unsigned short i=0;i<p.getNumTraits();++i){
       adGenVal[i]+=lit->getLocusEffect(i);
     }
      }
    }
```

```
  }

-----------------------------------------------------------------

  void Individual::calcPhenVal(Parameters& p){
    for(unsigned short i=0;i<p.getNumTraits();++i){
      envDevVal[i] = gsl_ran_gaussian(prand,
sqrt(p.getGenAdCov(i,i)*(1/p.getHeritability(i)-1)));
//generating environment deviates;
      phenVal[i]=p.getTraitMeans(i)+adGenVal[i]+envDevVal[i];
    }
  }

-----------------------------------------------------------------

  char Individual::pickGender(){
    double rNumber=gsl_rng_uniform(prand);
    if(rNumber<0.5)
      return 'f';
    else
      return 'm';
  }

-----------------------------------------------------------------

  void Individual::zygogenesis(Parameters& p,Individual&
s,Individual& d){

//*************************************************************
//*Sampling a chromosome in the individual's parents-BEGINNING *
//*************************************************************
    unsigned short sIndex=0;
    unsigned short dIndex=0;

    double sNumber=gsl_rng_uniform(prand);
    if(sNumber<0.5)
      sIndex=1;

    double dNumber=gsl_rng_uniform(prand);
    if(dNumber<0.5)
      dIndex=1;

//*********************************************************
//*Sampling a chromosome in the individual's parents-END *
//*********************************************************

    vector<Chromosome>::iterator icit;
```

```
    vector<Chromosome>::iterator scit;
    vector<Chromosome>::iterator dcit;
    vector<Locus>::iterator ilit;
    vector<Locus>::iterator slit;
    vector<Locus>::iterator dlit;

    double rbNumber=0;


for(icit=genome.getGenom().begin(),scit=s.genome.getGenom().begi
n(),dcit=d.genome.getGenom().begin();icit!=genome.getGenom().end
(),scit!=s.genome.getGenom().end(),dcit!=d.genome.getGenom().end
();++icit,++scit,++dcit){
      for(ilit=icit->getChrom().begin(),slit=scit-
>getChrom().begin(),dlit=dcit->getChrom().begin();ilit!=icit-
>getChrom().end(),slit!=scit->getChrom().end(),dlit!=dcit-
>getChrom().end();++ilit,++slit,++dlit){

//*************************************************************
//*Test and execute recombination for sire's chromosomes-
BEGINNING  *
//*************************************************************
     rbNumber=gsl_rng_uniform(prand);
     if(rbNumber<p.getRecombRate()){
       if(sIndex==0)
         sIndex=1;
       else
         sIndex=0;
     }
//*************************************************************
//*Test and execute recombination for sire's chromosomes-END   *
//*************************************************************


//*************************************************************
//*Test and execute recombination for dam's chromosomes-
BEGINNING   *
//*************************************************************
     rbNumber=gsl_rng_uniform(prand);
     if(rbNumber<p.getRecombRate()){
       if(dIndex==0)
         dIndex=1;
       else
         dIndex=0;
     }
//*************************************************************
//*Testing and executing recombination for dam's chromosomes-END
```

```
//*************************************************************

     ilit->setAlellesConfig(0,1,slit-
>getAlellesConfig(sIndex),dlit->getAlellesConfig(dIndex));
     ilit->setTraitId(slit->getTraitId());
//It is needed because the method setLocusEffects requires that
traitId variable have been set previously. traitId could be
copied from sire or dam.
     ilit->setLocusEffect(p);
      }
    }
  }
}
-------------------------------------------------------------------
-------------------------------------------------------------------
```